Jassu Ilama

# Dynaaminen teksturointi reaaliaikaisessa 3D-grafiikassa

Metropolia Ammattikorkeakoulu

Medianomi

Viestintä

Opinnäytetyö

30.4.2014

Metropolia

| | |
|---|---|
| Tekijä(t)<br>Otsikko<br><br>Sivumäärä<br>Aika | Jassu Ilama<br>Dynaaminen teksturointi reaaliaikaisessa 3D-grafiikassa<br><br>36 sivua + 1 liite<br>30.4.2014 |
| Tutkinto | Medianomi |
| Koulutusohjelma | Viestintä |
| Suuntautumisvaihtoehto | 3D-Animointi ja visualisointi |
| Ohjaaja(t) | Jaro Lehtonen |


Opinnäytetyön aiheena oli tutkia ja kehittää täysin uusi dynaaminen teksturointimenetelmä reaaliaikaiseen 3d-grafiikkaan käyttäen Unreal Development Kitin työkaluja. UDK:n materiaalieditori antaa oivan alustan varjostimien kehittämiselle ilman perinteistä koodaamista. Varjostimien visuaalinen koodausympäristö antaa mahdollisuuden tehdä prototyyppejä varjostimista äärimmäisen nopeasti.

Työssä tutkitaan pääsijaisesti kahta menetelmään liittyvää varjostinohjelmaa, monitekstuurisysteemiä ja loputtomasti jatkuvia tekstuureja, ja niiden yhdistämistä. Menetelmän käyttämistä testataan esimerkkitapauksessa, jossa käsiaseen teksturoiminen toteutetaan kokonaan käyttämällä kehittämääni menetelmää.

Menetelmä perustuu värialuekarttaan, josta voidaan erotella kuusi eri maskialuetta. Näitä maskitekstuureja voidaan käyttää uniikkien toistuvien tekstuureiden asettamiseen värialuekartan määräämiin alueisiin samanaikaisesti. Menetelmää hyödynnetään materiaali-instansseissa, joissa materiaalien muokkaaminen toimii reaaliaikaisesti suoraan pelimoottorissa.

Työssä käydään lyhyesti läpi myös 3d-mallinnus-, uv-kartoitus ja teksturointiosuus työssä esiintyvälle käsiaseelle, ja tehdään pientä vertailua Borderlands pelin materiaalisysteemiin.

Työssä kuvatut menetelmät voivat antaa uusia näkökulmia varjostinohjelmien rakentamiseen kokeneellekin grafiikkaohjelmoijalle.



| | |
|---|---|
| Avainsanat | udk, varjostin, materiaali, maski, matemaattinen, 3d, grafiikka, mallinnus, teksturointi, uv-kartoitus, peli, reaaliaika |

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages<br>Date | Jassu Ilama<br>Dynamic texturing in real-time 3D graphics<br><br>36 pages + 1 appendix<br>30 April 2014 |
| Degree | Bachelor of Media |
| Degree Programme | Culture |
| Specialisation option | 3D Animation and Visualization |
| Instructor(s) | Jaro Lehtonen |

The objective of this thesis was to research and manufacture a completely new way of doing dynamic texturing in real-time 3d graphics using the tools of the Unreal Development Kit. The material editor of UDK gives a great advantage to graphics programming with the lack of traditional text based programming. The visual coding environment gives the opportunity to make prototypes of shaders very quickly.

The aspects primarily examined in this thesis are the two main shader programs. One of the shader programs separates mask textures from a multicolored texture, and the other program handles the tiling of textures in a material library texture. These systems are then combined to create a material, where all the individual masked colors of the multicolored texture are filled with unique tileable textures. This system is then used in an example that explains how it can be used in practice with the complete texturing of a handgun.

The modeling, UV-unwrapping and texture baking is also briefly explained. Also, the method is also briefly compared against the system used in the Borderlands videogame, explaining the pros and cons of both systems.

The described methods of this thesis can give new insight to graphics and shader programming even for a graphics programming expert.

| | |
|---|---|
| Keywords | udk, shader, material, mask, math, 3d, graphics, modeling, texturing, uv-unwrapping, game, real-time |

**Sisällys**

## Lyhenteet ja termit

| | |
|---|---|
| Shader / Varjostin | Varjostinohjelma, joka sisältää ohjeet näytönohjaimella laskettavan materiaalin piirtämiseen 3d-moottorissa |
| Materiaali | Yhdistelmä tekstuureja ja matemaattisia varjostinohjelmia |
| Tekstuuri | 3d-mallien materiaaleissa käytettävä bittikarttakuva |
| Normaalikartta | Bittikarttakuva, joka sisältää 3d-malliin viittaavaa geometrista dataa, tätä tekstuuria hyödyntäen voidaan tehdä esimerkiksi low-poly mallista high-poly mallin näköinen |
| Spekulaarikartta | Tekstuurikartta, joka simuloi valon heijastuksia 3d-mallin pinnalla |
| Diffuusikartta | Pelkästään pinnan väri-informaatiota sisältävä tekstuurikartta |
| Emissiokartta | Tekstuuri, joka pitää sisällään dataa, jota käytetään pintojen valoissa ja valaisemisessa. |
| Beikkaus / leivonta | Esilaskenta, esimerkiksi dynaamiset varjot voidaan laskea suoraan tekstuuriin pintaan keventäen varjomoottorin työtä pelimoottorissa |
| Lerp | Linear Interpolation, kahden värin yhdistäminen käyttämällä mustavalkoista maskia |
| Frac | Matemaattisen arvon toistaja |
| Ambient occlusion | Yleisvalaistusta simuloiva jälikäsittelyefekti |
| Polygoni | Monikulmio, 3d-mallit rakentuvat kolmioista ja monikulmioista |

# 1 Johdanto

Teksturointitekniikat ovat pysyneet pitkään samoina, ja vain kourallinen pelien kehittäjistä on uskaltanut kehittää ja kokeilla uusia menetelmiä. Hyvänä esimerkkinä ID Softwaren RAGE videopeli, jonka teksturoimisessa käytetty ”megatexture” antoi pelille mukavan annoksen lisää uskottavuutta ja realismia.

Aihe tuli mieleen tutkiessa modulaarisia ratkaisuja tietokonepelien aseiden rakentamisessa. Modulaarisuus on ollut minulle kiinnostuksen kohteena jo pidemmän aikaa, koska sen hyötyjä tietokonepelien tekemisessä on monia. Esimerkiksi kenttäsuunnittelussa voidaan hyödyntää modulaarisia palasia käytävien ja peliympäristöjen rakentamisessa.

Keskityn kuitenkin materiaalien ja pintojen teksturointiin, ja niiden antamiin mahdollisuuksiin ja haasteisiin. Dynaamisia materiaalisysteemejä ei juurikaan käytetä nykypäivän peleissä. Dynaaminen modulaarisuus teksturoinnissa voisi antaa uusia näkökulmia pelien kehittämiseen ja avartaa nykyisiä menetelmiä. Esimerkiksi videopeleissä esiintyvien aseiden materiaaleja voitaisiin dynaamisesti muuttaa reaaliaikaisesti pelimoottorin sisällä. Tämä antaisi mahdollisuuden tehdä 3d-mallien pinnoilla esiintyvistä tekstuureista rajattomasti variaatioita erittäin nopeasti ja helposti.

Dynaamisesti muokattavat tekstuuripinnat voivat antaa uusia ideoita reaaliaikaisiin materiaalisysteemeihin. Kyseistä tekniikkaa ei kuitenkaan olla viety kovinkaan pitkälle tänä päivänä. Peliteollisuudessa dynaaminen materiaalisysteemi voisi helpottaa tekstuuriartistien työtä, ja olla hyödyksi myös pelin graafisen ilmeen yhdenmukaisuuden säilyttämisessä.

Kerron aluksi eri käsitteistä ja työkaluista, ja kuinka niitä yleensä käytetään 3d-grafiikassa, tästä siirryn oman prosessin kuvaukseen ja purkuun, ja lopuksi teen pientä vertailua Gearboxin Borderlands pelin materiaalisysteemiin. Työssä ei keskitytä pääsijaisesti mallintamiseen, joten mallintamiseen liittyvät asiat käydään läpi melko lyhyesti ja suurpiirteisesti.

# 2 Nykypäivän 3D- tekniikat ja työkalut

Tässä osiossa kerrotaan asioita liittyen perinteisiin mallinnus- ja teksturointitekniikkoihin, uv-kartoitukseen ja varjostinohjelmiin.

## 2.1 3d- mallit ja mallinnus

3d-mallit edustavat 3d-objektia käyttäen kokoelmaa pisteitä 3d-avaruudessa, joita yhdistävät erilaiset geometriset yksiköt, kuten kolmiot, viivat, kaarevat pinnat jne. Koska mallit ovat kokoelma dataa (pisteitä ja muuta informaatiota), 3d-malleja voidaan luoda käsin, tietokonealgoritmeilla tai skannaamalla. (wikipedia 2014a.)

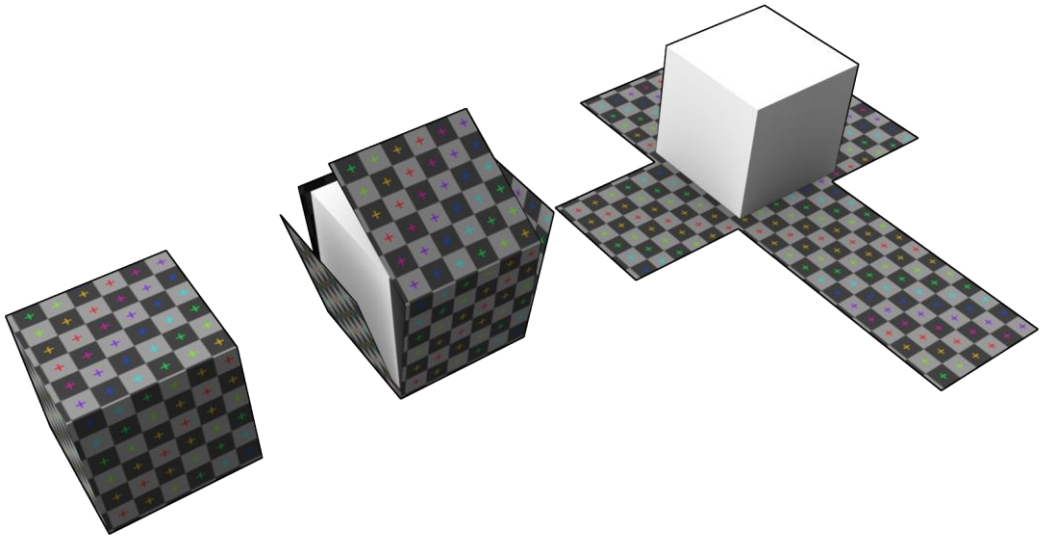
3d-malleja käytetään laajasti kaikkialla 3d-grafiikassa. Oikeastaan niiden käyttö edelsi laajaa käyttöä 3d-grafiikassa tietokoneilla. Monet tietokonepelit käyttivät esirenderöityjä kuvia 3d-malleista spriteinä ennen kuin tietokoneet pystyivät renderöimään reaaliaikaisesti. (wikipedia 2014b.)

Polygonimallinnuksessa pisteet 3d-avaruudessa, joita kutsutaan vertekseiksi, on yhdistetty linjoilla, jotka muodostavat monikulmioisen pinnan. Valtaosa tämän päivän 3d-malleista on rakennettu teksturoituina polygonimalleina, koska ne ovat joustavia ja koska tietokoneet voivat renderöidä ne niin nopeasti. Kuitenkin polygonit ovat tasomaisia ja voivat tuottaa likimääräisen kaarevia pintoja vain käyttäen niihin enemmän polygoneja. (wikipedia 2014c.)

## 2.2 Uv- kartoitus

Uv-kartoitus on usein tärkeä alue 3d-grafiikkaa käyttävässä mediassa, niin peleissä, kuin elokuvateollisuudessakin.

Uv-kartoituksella määritetään 3d-ohjelmassa mallin pintaan alueet, joiden sisään 2d-tekstuuri asettuu. Tämä prosessi projisoi tekstuurikartan 3d- objektin päälle. Kirjaimet ”U” ja ”V” tarkoittavat akseleita 2d-tekstuurissa, koska ”X” ja ”Y” akselit ovat jo käytössä 3d-objektin mallitilassa. UV-teksturointi mahdollistaa polygonien maalaamisen kuvatiedostolla.



*Kuva 1. Representaatio kuution uv-alueen kartoittamisesta. (wikipedia 2014d)*

3d-mallin pinnoilla on muoto, koko, sijainti avaruudessa ja jopa paikallinen orientaatio. Yksikään näistä parametreista ei kuitenkaan auta meitä määrittelemään, kuinka tekstuurit piirretään näiden pintojen pinnalle. Tätä varten tarvitsemme järjestelmän, joka on tarkoitettu kartoittamaan tekstuuridataa verkon pinnalle. Tämän vuoksi keksittiin UV-kartoitus. Kartoitusta voidaan ajatella kuvitteellisena kaksi- tai kolmiulotteisena muotona, jonka päälle tekstuuri projisoidaan ennen kuin sitä sovelletaan itse verkkoon. Kun uv-kartoitus on määritetty, sitä voidaan skaalata, pyörittää ja editoida verkon muodosta riippumatta. Tällöin tapaa, jolla tekstuuri piirretään varsinaiselle verkkopinnalle muutetaan. (Booktype 2014.)

## 2.3 Teksturointitekniikoita

Teksturointitekniikoita on monia, mutta tänä päivänä useimmin käytettyjä ovat todennäköisesti esilasketut, ”leivotut” tekstuurit. Tekstuurien leipominen high-poly mallista low-poly malliin on äärimmäisen kätevä tapa saada low-poly-mallin pintaan lisää yksityiskohtia suoraan high-poly-mallin tarkasta geometriasta. Näitä tekstuureja voi olla esimerkiksi ambient occlusion, joka pitää sisällään yleisvalaistukseen liittyvää dataa mustavalkoisessa tekstuurissa, tai normaalikartta, joka pitää sisällään tarkan representaation high-poly-mallin geometriasta 2d-bittikarttana.

Teksturointia tehdään myös hyvin paljon manuaalisesti kuvankäsittelyohjelmissa usein, koska leipomisella ei aina saada täydellistä lopputulosta. Photoshop on oiva työkalu tekstuurien muokkaukseen, ja sitä on perinteisesti käytetty 3d-mallien manuaaliseen teksturointiin. Muita suhteellisen uusia teksturointiin soveltuvia ohjelmia ovat esimerkiksi Pixologicin Z-Brushin ja Autodeskin Mudbox, jokta antavat mahdollisuuden maalata tekstuuria suoraan 3d-mallin pintaan ja samalla suoraan uv-kartalle tekstuurina.

## 2.4 Varjostinohjelmat

Varjostinohjelmat ovat usein tietokoneen näytönohjaimella laskettavia matemaattisia ohjelmia, jotka operoivat pikselien ja / tai verteksien tasolla. Varjostinohjelmilla voidaan luoda monimutkaisia efektejä, esimerkiksi reaaliaikaisia heijastuksia, vääristyksiä tai varjostuksia. Varjostimia on käytetty peleissä jo pidemmän aikaa, koska ne tarjoavat paljon mahdollisuuksia tehdä 3d-maisemista ja kappaleista entistäkin hienompia ja realistisempia. Varjostinohjelmilla yritetään usein huijata oikeassa maailmassa tapahtuvaa valon käyttäytymistä realistisien lopputulosten saavuttamiseksi.

Varjostinohjelmat laskevat renderöitäviä efektejä näytönohjaimella korkealla joustavuudella. Useimmat varjostinohjelmat ovat koodattu grafiikanprosessointiyksiköllе (GPU:lle), vaikka tämä ei ole tiukka vaatimus. Asentoa, värisävyä, värikylläisyyttä, kirkkautta ja kontrastia kaikissa pikseleissä, vertekseissä tai tekstuureissa, joita käytetään lopullisen kuvan rakentamisessa, voidaan muuttaa lennossa käyttäen algoritmeja, jotka on määritelty varjostinohjelmassa. (wikipedia 2014e.)

# 3 Työkalut ja työvaiheet

Tässä osiossa kerron lyhyesti työkaluista ja vaiheista liittyen valmiin 3d-mallin ja tekstuureiden tuottamiseen.

## 3.1 Blender

Blender on ilmainen vapaan lähdekoodin 3d-grafiikka ja mallinnusohjelma, jota käytetään mm. animaatioelokuviin, visuaalisiin efekteihin, taiteeseen, printattuihin 3d-malleihin, UV-kartoittamiseen, teksturointiin, rigaukseen, skinnaukseen, kompositointiin, videoeditointiin, ja monenlaisiin simulaatioihin. Kaikkien mallinnustyökalujen lisäksi Blenderissä on sisäänrakennettu pelimoottori. (wikipedia 2014f)

Blender oli luonnollinen valinta 3d-osuuden tuottamiseen aikaisemman kokemuksen vuoksi. Blenderin työkalujen nopea kehitys on vakiinnuttanut ohjelman jo monien pelinkehittäjien parissa.

### 3.1.1 Mallinnus

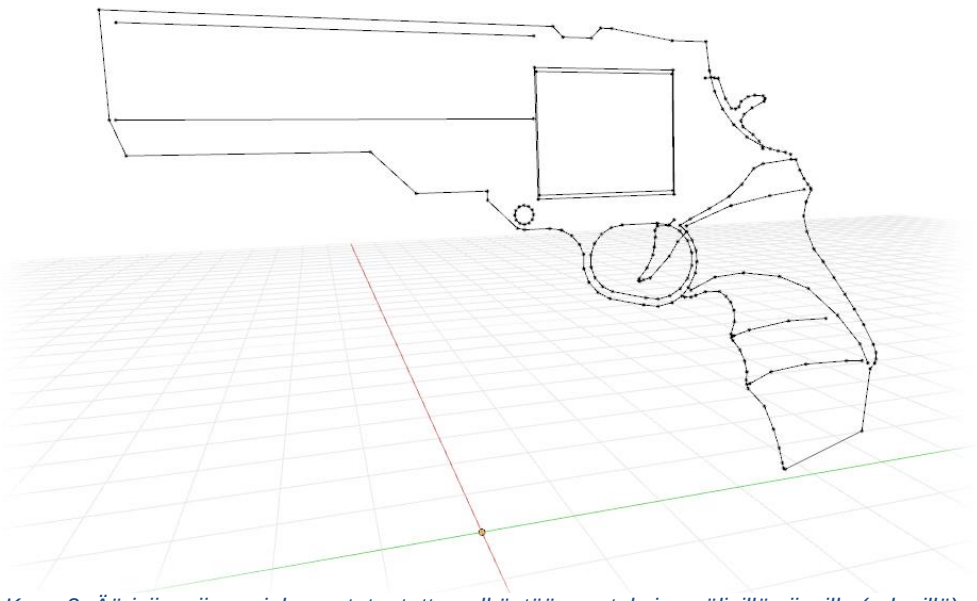
Suunnitellessani objektia, jolla pystyisin hyvin demonstroimaan materiaalisysteemin joustavaa toimintaa. Yritin miettiä oikeasta maailmasta löytyviä esineitä, joista löytyy niin mekaanisia kuin orgaanisiakin muotoja.

Vaihtoehdoiksi karsin kolme mallia: osittain robottimaisen ihmismallin, käsiaseen ja melko villin muotoisen espressokeittimen. Päädyin kuitenkin valitsemaan käsiaseen, koska siinä oli riittävästi mallinnettavaa, ja pystyisin silti hyödyntämään koko teksturointimenetelmää sen kanssa.

Ihmismallin suunnitteluun olisi helposti voinut kulua liikaa aikaa, ja sen mallintamiseen sitäkin enemmän. Espressokeittimestä olisi löytynyt kaikki halutut muodot, mutta ajattelin kuitenkin, että yleensä peleissä espressokeittimillä on usein pienempi rooli kuin itse peleissä esiintyvillä aseilla.

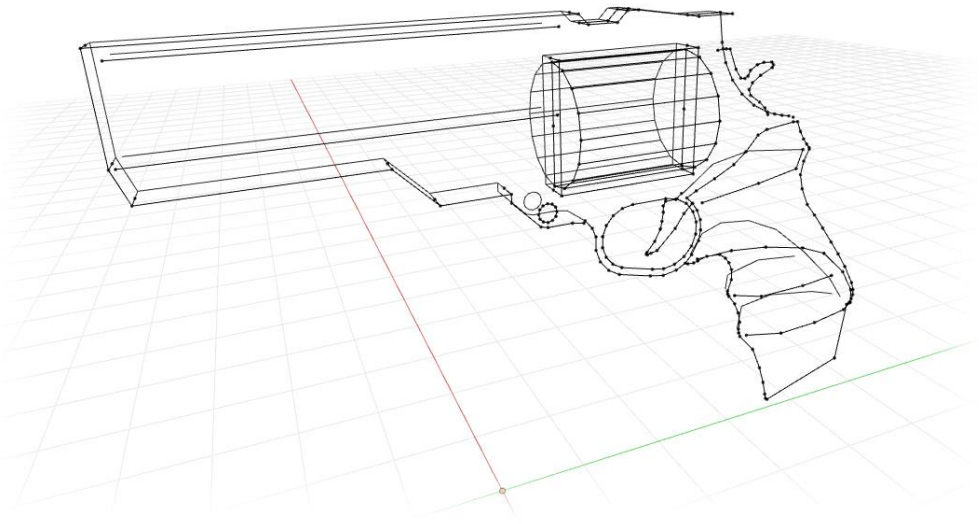
3d-Mallinnus on usein vaihe, joka tulee vasta paperille suunnittelun tai referenssimateriaalin tutkimisen jälkeen. Omassa tapauksessani lähdin työstämään suunnitelmaa suoraan Blenderin sisällä, enkä hahmotellut tai suunnitellut paperille mitään.

Olen kehittänyt itselleni tekniikan, jota olen käyttänyt lähes jokaisessa mallintamassani esineessä sen oivallettuani. Tämä tekniikka perustuu pelkästään verteksien välille syntyviin viivoihin (edge) ja verteksien työntämiseen 3d-avaruudessa. Suunnitteluvaiheessa en luo peittäviä pintoja yleensä lainkaan, vaan pidän kappaleen kokonaan lankamallina. Piirtämällä verteksiviivoilla aseen ääriviivat ja tärkeimmät yksityiskohdat ensin sivusta saadaan selkeä 2d-piirrosmainen kuva.



*Kuva 2. Ääriviivapiirros, joka on toteutettu pelkästään verteksien välisillä viivoilla (edgeillä).*

Ääriviivojen piirtämisen jälkeen pystytään helposti työntämään lisää verteksiviivoja volyymialueiden reunoille antaen riittävän kolmiulotteisen vaikutelman mallista. Käyttäen peilausmuokkainta (mirror modifier) saadaan verteksiviivat mallin molemmille puolille samanaikaisesti säästäen aikaa ja työtä.



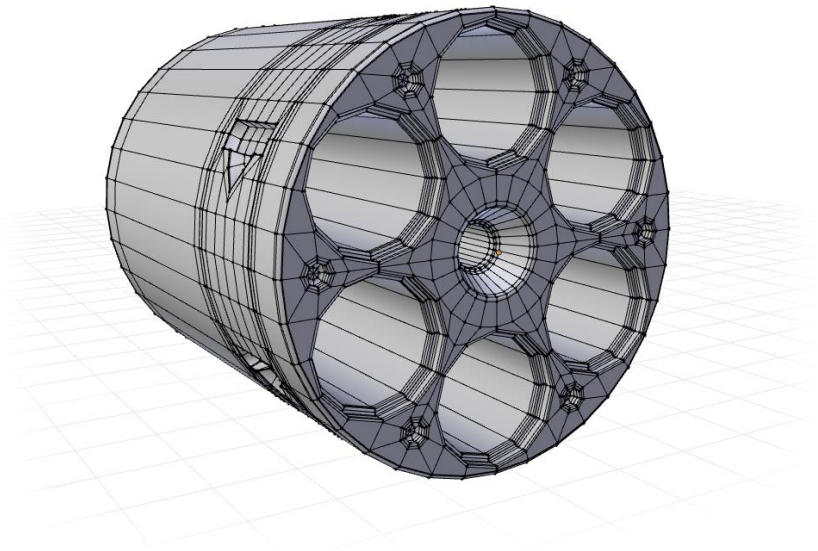
*Kuva 3. Malliin saavutettu syvyysvaikutelma verteksien kopioimisella ja niiden pienellä liikuttelulla.*

Tämä työvaihe kestää yleensä vain muutaman hetken, ja malli on näin edelleen erittäin helposti muokattavissa ja silti riittävän kolmiulotteinen hahmottamista varten.

Tekniikan suurimmat edut ovat nopeassa editointimahdollisuudessa ja riittävässä volyymin visualisoimisessa.
Haittapuolena on, että tätä tekniikkaa ei pystytä käyttämään kaikissa 3d-mallinnusohjelmissa niistä johtuvien rajoitusten vuoksi.

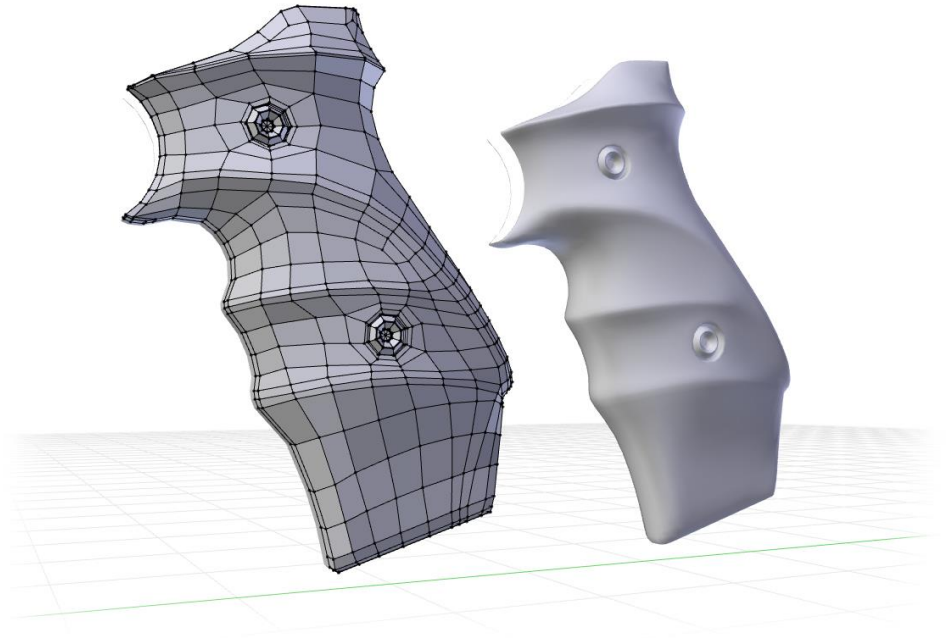
Verrattuna perinteiseen paperille suunnitteluun, volyymin hahmottaminen 2d-piirtämisessä voi olla vaikeaa suunnitteluvaiheessa, mutta antaa mahdollisuuden väritykselle, ja tyylittelylle. Myös 2d-suunnitelmapiirroksen muokkaus voi olla usein työlästä, ja kuvakulma on aina rajoitettu.

Itse mallinnus- ja pintojentäyttövaiheessa täytyi kiinnittää huomiota geometriaan, jotta säästyttäisiin ylimääräisiltä venymisiltä ja vääristymisiltä.



*Kuva 4. Tarkasti mallinnetut yksityiskohdat lisäävät uskottavuutta normaalikarttoja leipoessa.*

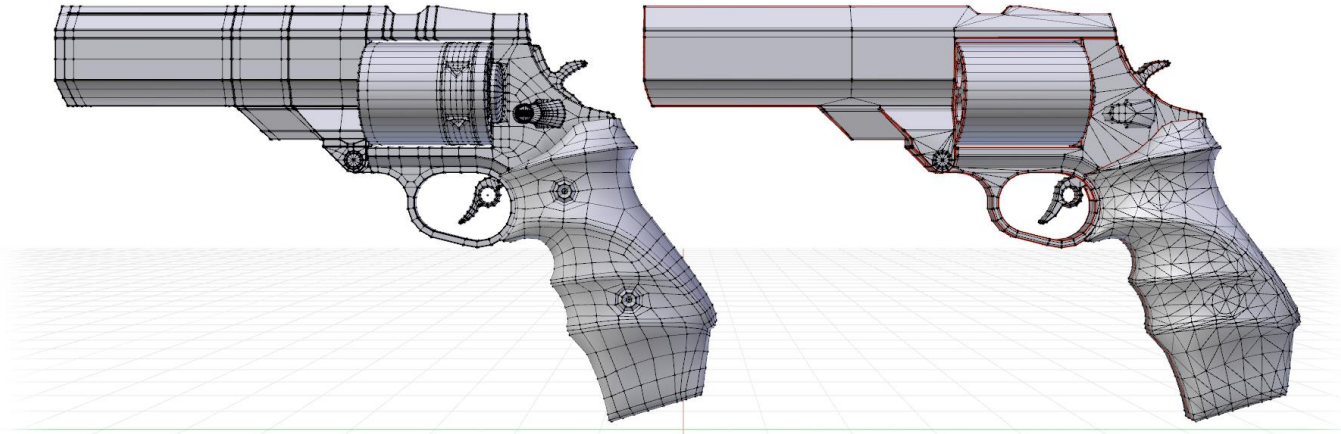
High-polyn mallintamisessa yritin saada mahdollisimman tarkasti yksityiskohtia rakennettua, jotta saisin jokseenkin uskottavan näköisen lopputuloksen. Käyttämällä subsurf-muokkainta (subsurf modifier) pystytään helposti lisäämään malliin tarkkuutta ja pehmentämään mallintaessa syntyviä teräviä kulmia. Kulmat, jotka halutaan pysyvän teräviä vaativat usein lisää geometriaa, esimerkiksi kulman molemmille sivuille voidaan lisätä ylimääräinen edgeluuppi lähelle taitospistettä. Tällöin subsurf-muokkaimelle jätetään vähemmän tilaa pyöristyksen laskemiseen.



*Kuva 5. Valmis kahva editointi moodissa (vas.) ja subsurf-muokkaimen kanssa (oik).*

Pidin myös mallinnusvaiheessa mielessä seikkoja, jotka vaikuttivat myöhemmin low-poly-mallin rakentamiseen. Esimerkiksi alkuperäisen geometrian säilyttäminen subsurf-muokkaimen alla helpotti low-poly-mallin rakennusvaiheessa.

Low-poly-malli oli suhteellisen nopeasti valmiiksi rakennettu suoraan high-poly-mallista. Pystyin helposti poistamaan subsurf-muokkaimen, joka jätti työksi enää ylimääräisten nurkkiin asetettujen verteksiviivojen optimoimisen ja ylimääräisten yksityiskohtien poistamisen.



*Kuva 6. High-poly malli (vas.) ilman muokkaimia 28318 kolmiota, ja optimoitu low-poly malli (oik.) 8503 kolmiota.*

Mallin optimoimiseen olisin voinut käyttää paljon enemmän aikaa, mutta arvelin tämän olevan riittävä materiaali systeemin esittämiseen. Muutin myös kahvan orgaanisen pinnan kolmioiksi, koska usein pelimoottorit tai exportterit saattavat tehdä automaattisen kolmioinnin eri tavalla kuin itse mallinnusohjelma.

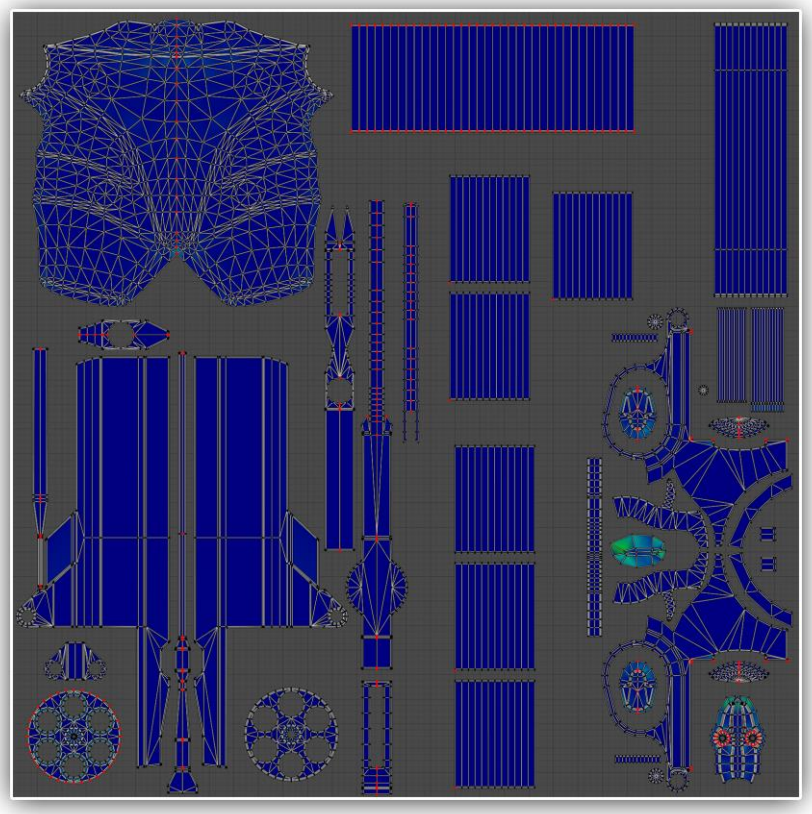
Peleissä optimoimiseen joudutaan usein kiinnittämään paljon enemmän huomiota, koska laitteistovaatimuksista johtuen ruudulla voidaan näyttää vain rajallinen määrä kolmioita samanaikaisesti.

### 3.1.2 UV- kartoitus

UV- kartoitusvaiheessa oli erittäin tärkeää pitää uv-saarekkeet 3d-mallin pintojen mukaisina, jotta säästyttäisiin turhilta venymisiltä leipomis- ja teksturointivaiheissa. Blenderin UV-työkalussa kätevänä asetuksena on venymisen visualisointi. UV-kartan jokainen pinta värjääntyy sinisen ja punaisen välille riippuen venymisen kriittisyydestä.

3d-mallin UV-saarekkeet yritetään järjestää karttaan optimoidusti jättäen käyttämätöntä tekstuuripintaa mahdollisimman vähän, mutta kuitenkin pitäen pienet marginaalit mahdollisia tekstuurivuotoja ajatellen.

Tästä uv-kartasta voidaankin päätellä, että pintaa jäi käyttämättä melko paljon.



*Kuva 7. Lopullinen uv-kartta. Sininen alue merkkaa täysin vääristymätöntä tekstuuripintaa.*

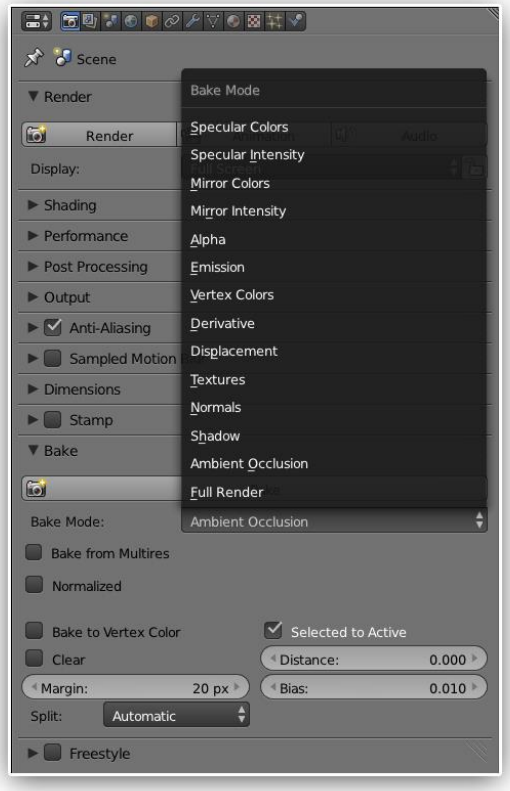
Koska tekstuurileipomiset tehtiin 4096*4096 pikseliresoluutiossa, tekstuuri tulee toistumaan riittävän kokoisena mallin päällä käyttämättömästä uv-tilasta riippumatta. Jos mallia käytettäisiin oikeassa pelituotantoympäristössä, voitaisiin uv-tilaa optimoida vielä paljon enemmän käyttäen esimerkiksi päällekkäisiä uv-saarekkeita, jotka voisivat käyttää samaa pohjalla olevaa tekstuuria.

*Kuva 8. Shakkiruututestitekstuuri auttaa tekstuurialueen suhteiden ja skaalan havainnollistamisessa.*

Uv-saarekkeiden järjestelyvaiheessa on olennaista pitää kaikki näkyvät pinnat mahdollisimman hyvin suhteutettuina. Jos osa uv-saarekkeista on suhteettoman kokoisia, voi pelimoottorin sisällä syntyä piirto-ongelmia saarekkeiden uv-saumoissa.

### 3.1.3 Tekstuurien leipominen Blenderissä (baking)

Tekstuurien leipominen Blenderissä on tehty erittäin helpoksi. Uv-kartoituksen jälkeen voidaan luoda uusi tekstuuri suoraan Blenderin sisällä haluttuihin parametreihin. Tekstuurin luomisessa voidaan muuttaa sen pikselikokoa, formaattia ja valita yksi useasta testitekstuurista. Testitekstuureita ovat esimerkiksi mustavalkoinen shakkiruutukuvio ja värillinen ruudukko. Näiden lisäksi voidaan asettaa tekstuurin sijaan haluttu täyttöväri.

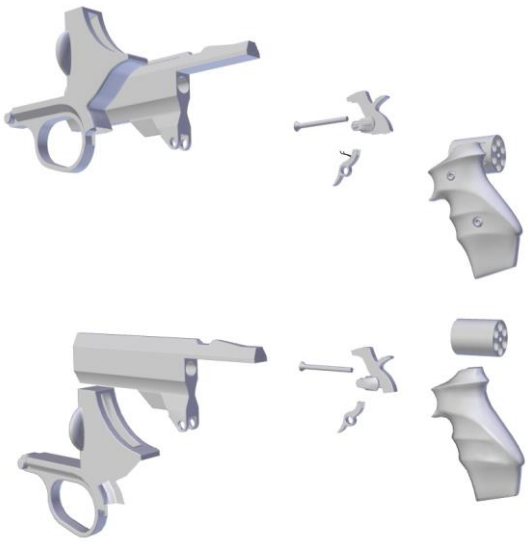



*Kuva 9. Kuvakaappaus Blenderin leipomistyökalusta, ja sen antamista leipomismahdollisuuksista.*

High-poly-mallin yksityiskohtien tallentaminen kuvaformaattiin on yksi käytetyimmistä menetelmistä nykypäivän 3d-peligrafiikassa. Tarkan mallin yksityiskohdat voivat antaa low-poly-mallille erittäin paljon lisää haluttua uskottavuutta.

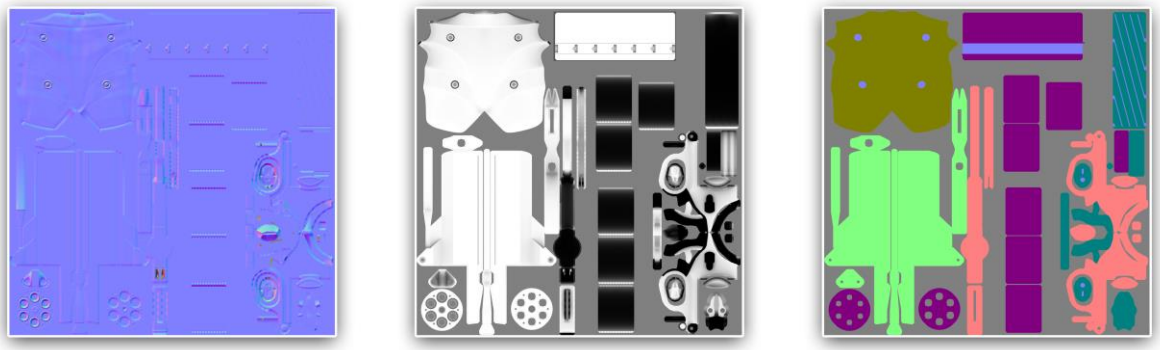
Omaan esimerkkiini tarvittavat leivokset olivat normaalikartta, ambient occlusion-kartta (yleisvalaistus ja varjokartta), ja hieman erikoisempi kustomoitu värialueita määrittävä kartta.

Mallin jakaminen pienempiin osiin auttaa tarkkojen tulosten saavuttamisessa leipomisvaiheessa. Mallin jaottelulla voidaan estää ongelmien ilmenemistä kappaleiden liitoskohdissa ja päällekkäin menevissä alueissa. Tämä toimenpide on hyvä tehdä vähintään normaalikarttoja leipoessa, riippuen tietenkin mallin geometriasta.



*Kuva 11. pienempiin osiin jaoteltu high- ja low-poly-malli.*

Normaaleissa stillkuvien renderöinnissä tietokoneen prosessori renderöi 3d-kameran näkemän alueen. Tekstuureja leivottaessa prosessori renderöi kuvan suoraan 3d-objektin uv-tekstuurikarttaan.



*Kuva 10. Valmiit leivotut tekstuurikartat. Vasemmalta oikealle: normaalikartta, ambient occlusion-kartta ja värialuekartta.*

## 3.2 Adobe Photoshop

Adobe Photoshop (usein myös pelkästään Photoshop) on Adobe Systemsin kehittämä kuvankäsittelyohjelma, joka on saavuttanut markkinajohtajuuden kaupallisessa digitaalisten kuvien muokkauksessa. Sovelluksen mainetta kuvastaa muun muassa se, että englannin kieleen on kehittynyt ohjelman nimeä muistuttava termi photoshopping (photoshoppaaminen), joka tarkoittaa yleisesti kuvan muokkaamista riippumatta käytetystä ohjelmasta. Monien muiden Adoben sovellusten tavoin Photoshop on saatavilla Microsoft Windowsille ja Mac OS:ille. (Wikipedia 2014g.)

Photoshop oli tuttuna kuvankäsittelyohjemana nopein tapa korjata ja editoida leivoksissa ilmeneviä ongelmia ja yksityiskohtia. Leivottuihin tekstuurikarttoihin voi usein ilmaantua pieniä artefakteja, jotka saattavat tulla näkyviin 3d-mallien teksturointivaiheessa. Normaalikarttoja tuodessa Blenderistä UDK:n contentbrowseriin, täytyy myös muistaa vihreän värikanavan kanavan invertointi.

Photoshopissa kasasin myös toistuvia tekstuureja sisältävän kirjastotiedoston, jota käytän myöhemmin toistuvien tekstuureiden varjostinohjelmaa rakentaessa.

## 3.3 UDK- pelimoottori

Unreal Engine -pelimoottori on pelimoottori, jonka on kehittänyt Epic Games. Tämä moottori oli ensimmäistä kertaa käytössä ensimmäisen persoonan ammuntapelissä nimeltä Unreal vuonna 1998. Vaikka moottori alun perin kehitettiin ensimmäisen persoonan ammuntapeleihin, sitä on käytetty onnistuneesti myös laajalti useissa eri genreissä, mukaan lukien hiippailupeleissä, massiivisissa roolimoninpelissä ja muissa roolipeleissä. Koska Unreal Enginen koodi on kirjoitettu C++ -ohjelmointikielellä, se mahdollistaa korkean siirrettävyyden muille alustoille, minkä vuoksi moottori onkin käytössä monilla pelinkehittäjillä tänäkin päivänä. (Wikipedia 2014h.)

Unreal Engine käyttää DirectX 9:n tuomaa HLSL- varjostinkieltä (High Level Shading Language), joka sallii erittäin monimutkaisten varjostinohjelmien koodaamisen.

Valitsin UDK-pelimoottorin, koska minulla oli jo hieman kokemusta kenttien, materiaalien ja logiikan rakentamisesta ohjelman sisällä. UDK:n kilpailijaan, Unityyn verrattuna, UDK:ssa on sisäänrakennettu rakennustyökalu, jolla varjostinohjelmia voidaan rakentaa helposti ja joustavasti vuokaavion tavoin. Käyttämällä valmiita matemaattisia funktionoodeja varjostimien koodaus onnistuu helposti jopa henkilöltä, jolla ei ole riittävää tietämystä perinteiseen varjostinohjelmien koodaamiseen. Koska UDK:n materiaalieditori mahdollistaa visuaalisen varjostinohjelmointikoodausympäristön, se toimii ponnahduslautana HLSL-varjostin kielen oppimiseen ja tehokkaaseen käyttämiseen.

### 3.3.1 Tekstuurien tuominen pelimoottoriin

Kuvia tuodessa UDK:n content browseriin on hyvä pitää mielessä, että kaikkien tuotavien kuvatiedostojen resoluutioiden täytyy olla kahden potenssissa, tai englanninkielellä ilmaistuna: ”the power of two”. Tekstuureja tuodessa content browseriin täytyy myös huomioida tekstuurien käyttötarkoitukseen liittyvät pakkausominaisuudet. Tuodessa tavallisia digitaalisia valokuvia ei asetuksiin juurikaan tarvitse koskea, mutta jos tuodaan normaalikarttoja tai maskeja, on asetettava tuontivaiheessa oikeat pakkausmoodit ja asetukset päälle, jotta tekstuurit toistuvat oikealla tavalla pelimoottorissa ja materiaalieditorissa. Esimerkiksi SRGB-väriavaruus täytyy usein kytkeä pois päältä maskitekstuureissa, jotta koko väriavaruutta pystytään tehokkaasti hyödyntämään matemaattisissa varjostinoperaatioissa.

### 3.3.2 3d-mallien tuominen pelimoottoriin

3d-mallien tuomisen Blenderistä content browseriin tein FBX -muodossa. Mallien tuomiseen liittyvä tärkeä asetus FBX-tuontiasetusikkunassa oli ’Explicit normals’, joka on vakiona kytketty pois päältä. Tämä tarkoittaa sitä, että 3d-mallin pintoihin tallennetut normaaliarvot lasketaan uudestaan UDK:n puolella. Tämän asetuksen pois päältä jättäminen usein aiheuttaa 3d-mallin päälle asetettujen normaalikarttojen vääristymisiä ja on siksi hyvä kytkeä aina päälle.

## 3.4 Unreal Material Editor

Materiaalieditorissa tapahtuu kaikkien pelimoottorissa esiintyvien materiaalien koodaus. Materiaalieditorilla voidaan koodata visuaalisesti noodipohjaisella käyttöliittymällä matemaattisia operaatioita ja efektejä. Materiaaleihin liittyviä varjostinohjelmia ovat esimerkiksi pikselivarjostimet, verteksivarjostimet ja geometriavarjostimet. Näillä ohjelmilla voidaan manipuloida 3d-mallin pinnalla olevia tekstuureja ja itse mallin geometriaa lähes äärettömällä eri tavalla. Materiaalieditorista saadaan myös visuaalisesti koodattu ohjelman raaka koodi pelkkänä tekstinä, joka voidaan kääntää lähes mihin tahansa pelimoottoriin, joka käyttää DirectX 9:ää.

Minulle visuaalinen koodausmenetelmä sopi mainiosti, koska minulla ei koodaushetkellä ollut vielä riittävää kokemusta perinteisestä tekstillä koodaamisesta. Pääsin heti jyvälle

matemaattisista operaatioista hyvin pitkälle sen takia, koska vastaava editori löytyy myös Blenderin materiaali- ja renderöintikompositointityökaluista.

# 4 Dynaaminen ja modulaarinen teksturointi

Tässä osiossa kuvataan materiaalivarjostimen suunnitteluvaiheesta ja itse systeemin toteuttamisesta materiaalieditorilla.

## 4.1 Haaste / Idea

Reaaliaikaisia teksturointimenetelmiä tutkiessa huomasin puutteita reaaliaikaiselle tekstuurimuokkaukselle. Tästä syntyi idea, jonka ajattelin olevan hyvä aihe lopputyölle.

Ideana oli saada tehtyä tekstuuri, josta pystytään matemaattisesti irrottamaan useampi mustavalkoinen maski reaaliajassa. Näitä maskeja pystyttäisiin sitten käyttämään eri materiaalipintojen kartoittamiseen, ja sen jälkeen täyttämään pinnat omilla tekstuureillaan dynaamisesti varjostinohjelmassa.

Tavoitteena oli kehittää työkalu, jonka avulla voidaan dynaamisesti teksturoida 3d-malleja käyttäen materiaalikirjastona toimivaa tekstuurikarttaa.

## 4.2 Suunnittelu ja moduulien toteutus

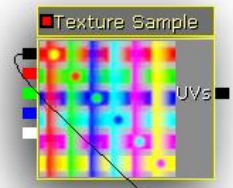
Osana työtä tutkin, että voitaisiinko harmaansävykartta käyttää värialuejaottelussa. Jakamalla maskitekstuurialueet eri harmaansävyiksi esimerkiksi seuraavilla harmaan valoisuusarvoilla 0 % (Musta), 10 %, 20 %, 30 %, … 90 %, 100 % (Valkoinen) pystytään nämä värialueet matemaattisesti erotella kymmeneksi erilliseksi maskiksi. Ongelmana tässä menetelmässä on kuitenkin se, että alueita ei pystytä sekoittamaan toisiinsa, ja jopa tekstuurin pakkaus UDK:n sisällä aiheuttaisi ongelmia. Pakkauksen aiheuttamat artefaktit saattavat sumentaa ja vääristää alueiden rajapintoja aiheuttaen visuaalisia virheitä 3d-mallin päällä. Myöskään reunanpehmennys (anti-aliasing) ei anna haluttua lopputulosta värialueiden pehmentämisessä, koska esimerkiksi täysin mustan ja valkoisen sekoittuminen tekee väkisin erisävyisiä harmaita pikseleitä.

Tämä menetelmä toimii sinänsä melko hyvin tarkoissa rajatuissa maskauksissa, mutta on äärimmäisen joustamaton pehmeiden materiaalirajojen saavuttamisessa, joten en kuluttanut aikaa tämän parissa sen enempää.

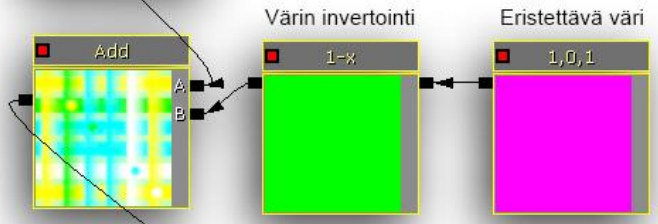
### 4.2.1 Matemaattinen värialuejaottelu maskeiksi

Yhdistetty RGB ja CMY-väripaletti.

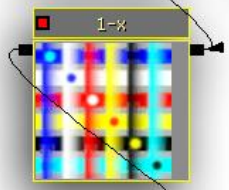
Ensimmäiset testaukset UDK:n materiaalieditorin sisällä aloitin RGB- ja CMY värit mielessä (Red, Green, Blue ja Cyan, Magenta, Yellow). Mietin matemaattisia operaatioita, joiden avulla pystyisin erottelemaan paletista määritetyllä haarukalla värejä erillisiksi maskeiksi. Pääsinkin varsin vaivattomasti haluttuun lopputulokseen. Alla selitän menetelmään liittyvät matemaattiset operaatiot käyttäen materiaalieditorista kuvakaapattua kaavaa.




Linkitetty bittikarttatekstuuri, joka on ladattu UDK:n content browseriin yksittäisenä PNG-tekstuurina. Tämän bittikartan ideana on tässä tapauksessa paljastaa eri värien sekoittumiseen liittyvät ongelmakohdat.

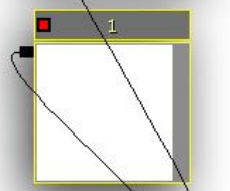



Eristettävän värin vastaväri täyttää RGB-kanavilla eristettävän värin täysin valkoiseksi.
Värin invertointioperaatiota on käytetty esimerkissä vain havainnollistamisen vuoksi, jo valmiiksi invertoitua väriä voitaisiin käyttää tässä tapauksessa.

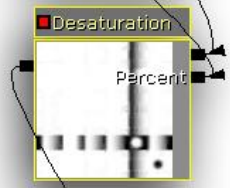



Kun eristetty väri on saatu valkoiseksi, se pystytään kääntämään vastaavalla invertointioperaatiolla täysin mustaksi.
Jos olisi käytetty vähennysoperaatiota (subtract), osa RGB-arvoista olisi saattanut mennä negatiivisiksi.
Alle nollan menevät RGB-arvot voivat käyttäytyvä joskus erittäin arvaamattomasti UDK:n sisällä.

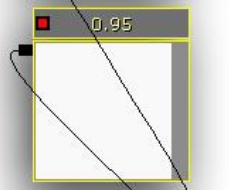

Vakiorajoitin leikkaa alle nollan menevät arvot tasan nollaksi, ja yli yhden menevät arvot tasan yhdeksi. Tällä operaatiolla saadaan tekstuurin RGB- arvot takaisin nollan ja yhden välille.



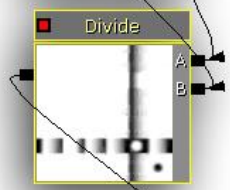

Arvo nollan ja yhden väliltä, joka desaturoimisen liittyvässä tapauksessa tarkoittaa, että nolla on nolla prosenttia, ja yksi on sata prosenttia desaturointia.



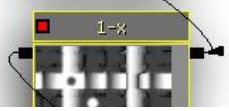

Värien desaturaatio-operaatio palauttaa tekstuurin värittömänä jättäen vain valoisuusarvot.

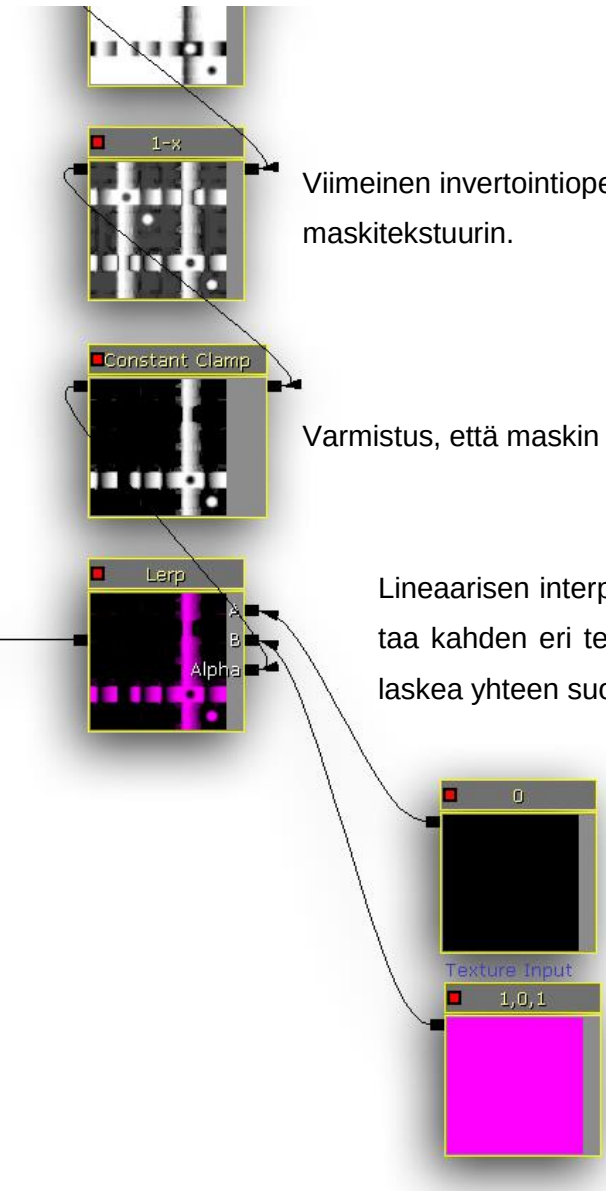



Arvo, jolla jaetaan.




Jako-operaatio arvolla 0,95 putsaa tekstuurin pakkauksesta tulleita artefakteja, nostamalla tekstuurin vaaleaa päätä. Tämä operaatio poistaa hieman maskin tarkkuutta, mutta kuitenkin hyväksyttävässä määrin.






Viimeinen invertointioperaatio palauttaa lähes lopullisen mustavalkoisen maskitekstuurin.

Varmistus, että maskin kaikki RGB-arvot ovat nollan ja yhden välillä.

Lineaarisen interpolaation alphakanavaan kytkettäessä maskitekstuuri mahdollistaa kahden eri tekstuurin saumattoman sekoittumisen. Tämä operaatio voidaan laskea yhteen suoraan muiden eristettyjen värialueiden kanssa.

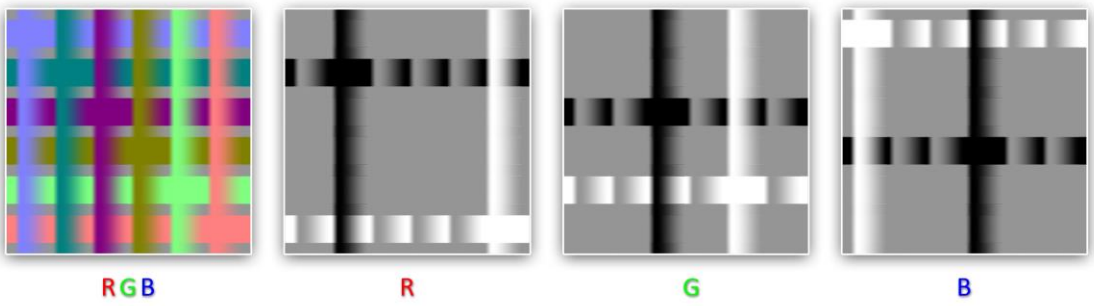
Jotta tätä ohjelmaa pystytään käyttämään muiden eristettyjen värialueiden yhdistämiseen, arvo pysyy tässä tapauksessa mustana.

Tämän värin tilalle pystytään kytkemään mikä tahansa haluttu tekstuuri, jonka halutaan näkyvän lopullisessa materiaalissa.

Tämä varjostinohjelma oli ensimmäinen soveltuvuusselvitys (”proof of concept”) siitä, että systeemi toimii hyvin pitkälle halutulla tavalla. Tämä ohjelma oli vasta ensimmäinen toteutettu versio. Halusin toimivuudesta huolimatta optimoida ohjelmaa kevyemmäksi nopeammin laskettavilla matemaattisilla operaatioilla ja järkevämmällä värialuekartalla.

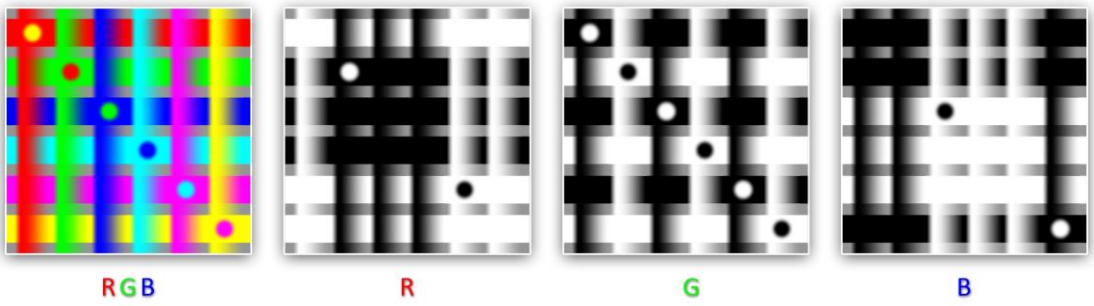
### 4.2.2 Lopulliset maskivärit värierottelussa

Vertailin ja testailin eri menetelmiä, kuinka yhdestä värillisestä bittikartasta saataisiin ulos mahdollisimman monta värikanavaa, jotka tukisivat sulavaa sekoittumista. Photoshopilla pystytään käsittelemään kuvia kaikilla RGB-kanavilla erikseen mustavalkoisessa formaatissa. Tämä antoi minulle idean, että jos asetan taustaksi keskiharmaan värin, silloin kaikilla RGB-kanavilla on sama väri. Pystyin täten maalaamaan suoraan esimerkiksi vihreälle kanavalle mustaa ja valkoista pehmeillä gradienteilla pitäen suurimman osan pinta-alasta silti keskiharmaana muita värikanavia varten. Jaottelin kuvan jokaiselle värikanavalle tietyn määrän mustaa ja valkoista.




*Kuva 12. Tekstuuri ja värikanavat eroteltuina.*

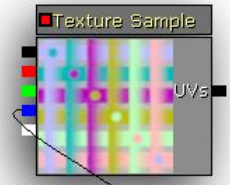
Värikanavia tutkittaessa voidaan havaita, että kaikilta kanavilta löytyy sekä musta että valkoinen värialue, jotka suoraan viittaavat vastaväripareihin. Nämä värialueet voidaan helpommin erotella toisistaan suoraan RGB-kanavien tasolla, koska yhdeltä kanavalta löytyy vain kaksi värialuetta. Vertauksena aikaisemmin käytetty väripaletti kanavineen.



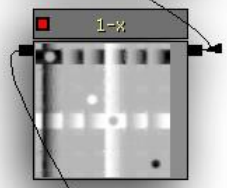

*Kuva 13. Kirkkaat värit ovat vaikeammin erotettavissa eri kanavilla.*

Kuvan [Kuva 13] RGB-kanavista voidaan huomata, että kaikki värialueet ovat näkyviä kaikilla värikanavilla, joka vaikeuttaa erittelyprosessia huomattavasti tapauksessani.

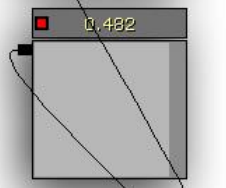
Uuden tekstuurin käyttäminen UDK:n materiaalieditorin sisällä on huomattavasti helpompaa suureksi osaksi kanavien erottelumahdollisuuden vuoksi. Oheisena on esimerkki tekstuurin käytöstä editorin sisällä.



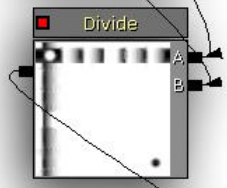

Linkitetty bittikarttatekstuuri, joka on ladattu UDK:n content browseriin erillisenä PNG-tekstuurina. Tämän bittikartan ideana on havainnollistaa eri värien sekoittumiseen liittyvät ongelmakohdat.
Aikaisempaan esimerkkiin verrattuna tekstuurista irrotetaan yhtä maskia varten vain sininen värikanava, joka on jo valmiiksi mustavalkoinen.



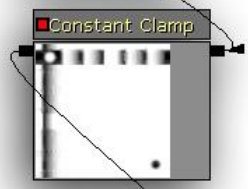

Lisäämällä invertointioperaatio pystytään valitsemaan keskiharmaan toisella puolella oleva valkoinen värialue. Ilman tätä vaihetta maskiksi saataisiin kanavan musta osuus.



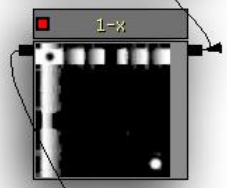

Jakamiseen käytettävä desimaaliarvo.



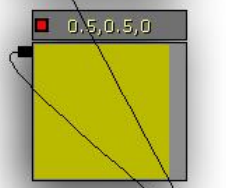

Jako-operaatio arvolla 0,482 siivoaa tekstuurin pakkauksesta tulleita artefakteja, ja samalla nostaa keskiharmaan värin valkoiseksi silti pitäen mustan mustana. Tällöin jo valkoisena oleva alue ylittää käytettävissä olevan alueen.



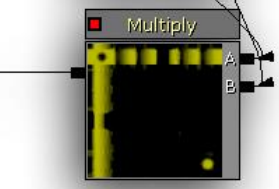

Vakioarvon rajaus 0-1 skaalaan muuttaa aikaisemmin yli yhden menevät arvot takaisin yhdeksi, jättäen ne täysin valkoisiksi.



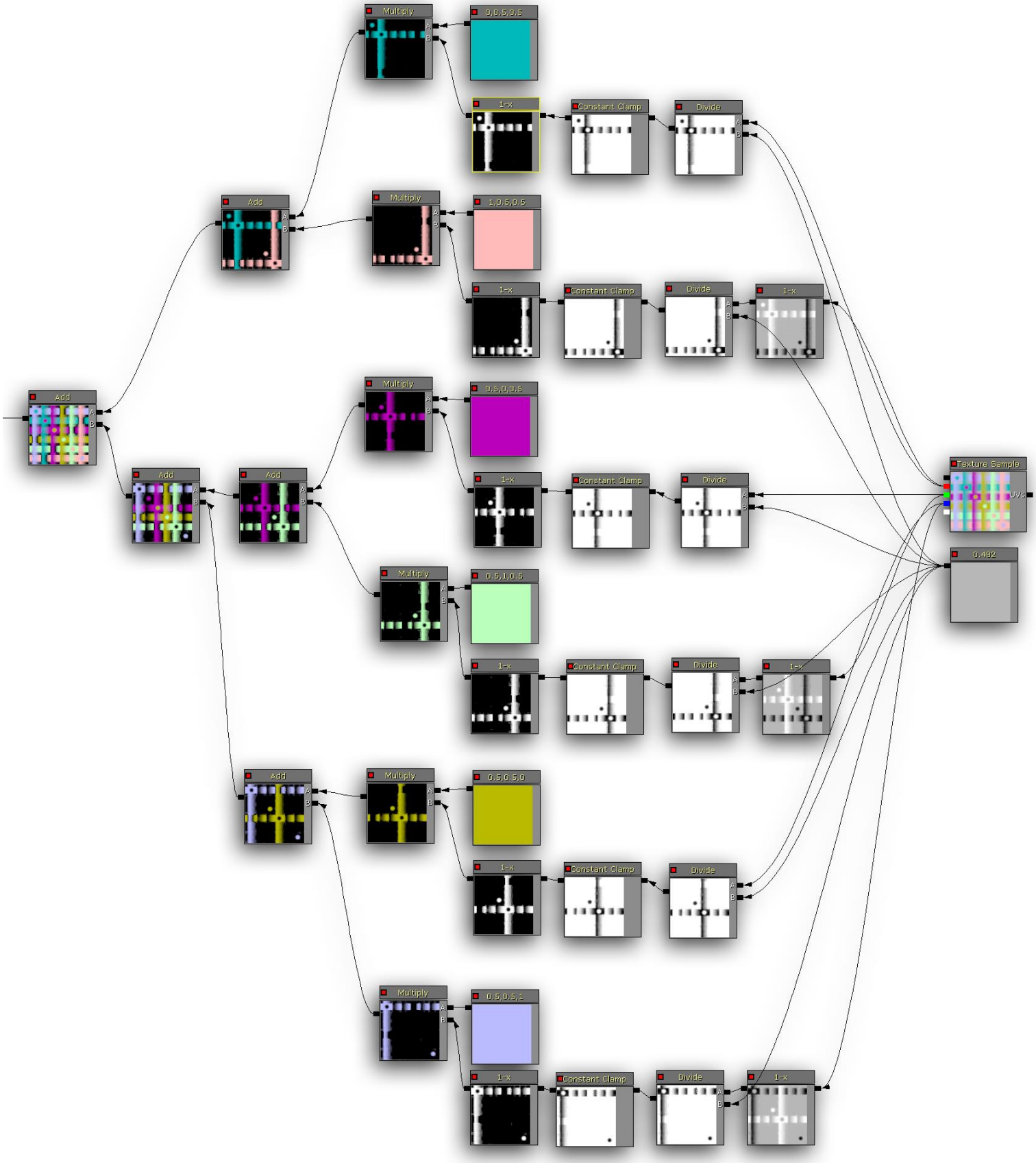

Viimeinen invertointioperaatio palauttaa lopullisen mustavalkoisen maskitekstuurin, jota voidaan käyttää toisen tekstuurin toistamiseen.




Tämän värin tilalle voidaan kytkeä mikä tahansa haluttu tekstuuri, jonka halutaan näkyvän lopullisessa materiaalissa maskin määrittämässä kohdassa.

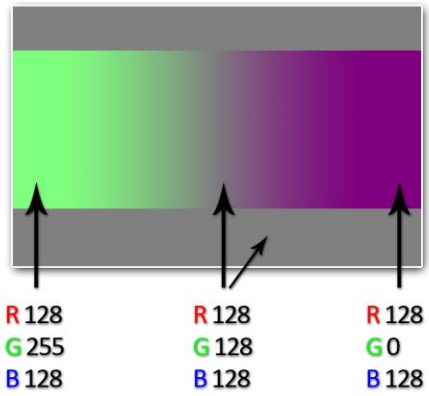



Toisin kuin aikaisemmin käytetty Linear Interpolate -operaatio, Multiply tekee tässä tapauksessa saman asian, mutta hieman pienemmällä laskumäärällä.

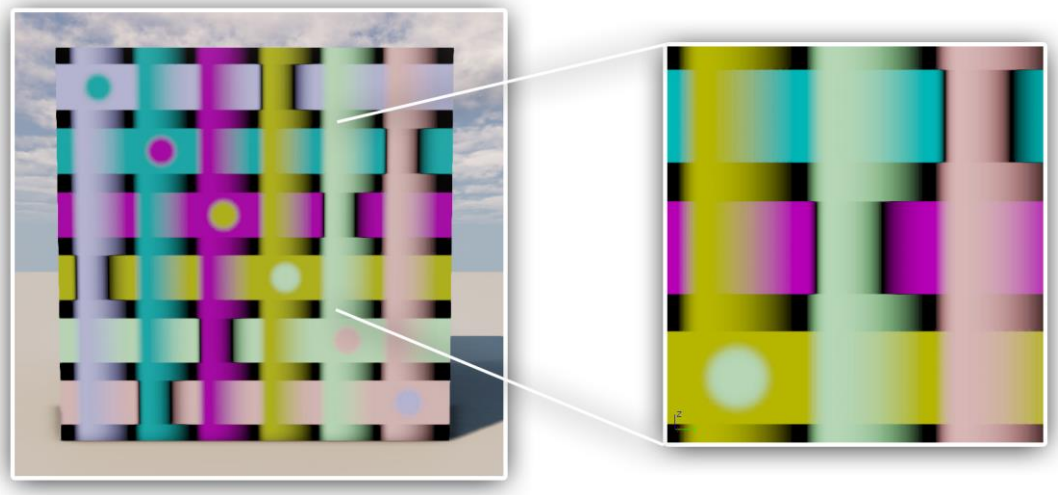



*Kuva 14. Kaikkien värialueiden erotteleminen ja taas takaisin yhdistäminen uusilla, mutta samoilla väreillä. Kaavan lukusuunta oikealta vasemmalle.*

Yhteenvetona väripaletti, johon lopulta päädyin pitää sisällään kuusi toisistaan helposti eroteltavaa väriä. Tämä paletti ei kuitenkaan takaa sataprosenttisen sulavaa sekoittumista kaikkien värialueiden kesken. Vastavärien sekoittuminen tuottaa edelleen niiden välille keskiharmaata, joka on ongelmallista värien erotteluvaiheessa. Vastavärien välille syntyvät harmaat alueet sekoittuvat taustan väriin, jolloin sitä ei pystytä tehokkaasti erottamaan varjostinohjelmalla.




*Kuva 16. Vastavärien sekoittuessa ilmenevä keskiharmaa alue.*



*Kuva 15. Vastavärien sekoittumisessa ilmenevä harmaa-alue ja sen aiheuttama ongelma materiaalin reaaliaikanäkymässä.*

Kyseinen ongelma ei kuitenkaan ole täysin ylitsepääsemätön. Tummaksi jääväiin kohtiin voidaan esimerkiksi yhdistää ambient occlusion-tekstuuri, jonka avulla voidaan peittää ongelmakohtia. Myös värialueiden merkkausvaiheessa voidaan välttää vastavärien sekoittumista ja näin eliminoida potentiaaliset ongelmakohdat. Kuvan esimerkissä kaikki värialueet sekoittuvat tarkoituksella toistensa kanssa, jotta nämä värien aiheuttamat ongelmakohdat voidaan helpommin havaita [Kuva 15].

## 4.3 Toistuvat tekstuurit

Toistuvia tekstuureja käytetään peleissä yleensä suurissa pinnoissa, esimerkiksi maastoissa. Toistuvat tekstuurit antavat mahdollisuuden värittää suuria pintoja käyttämättä suuria ja isoresoluutioisia tekstuureja, ja näin voidaan säästää kallisarvoista muistinkäyttöä pelimoottoreissa.

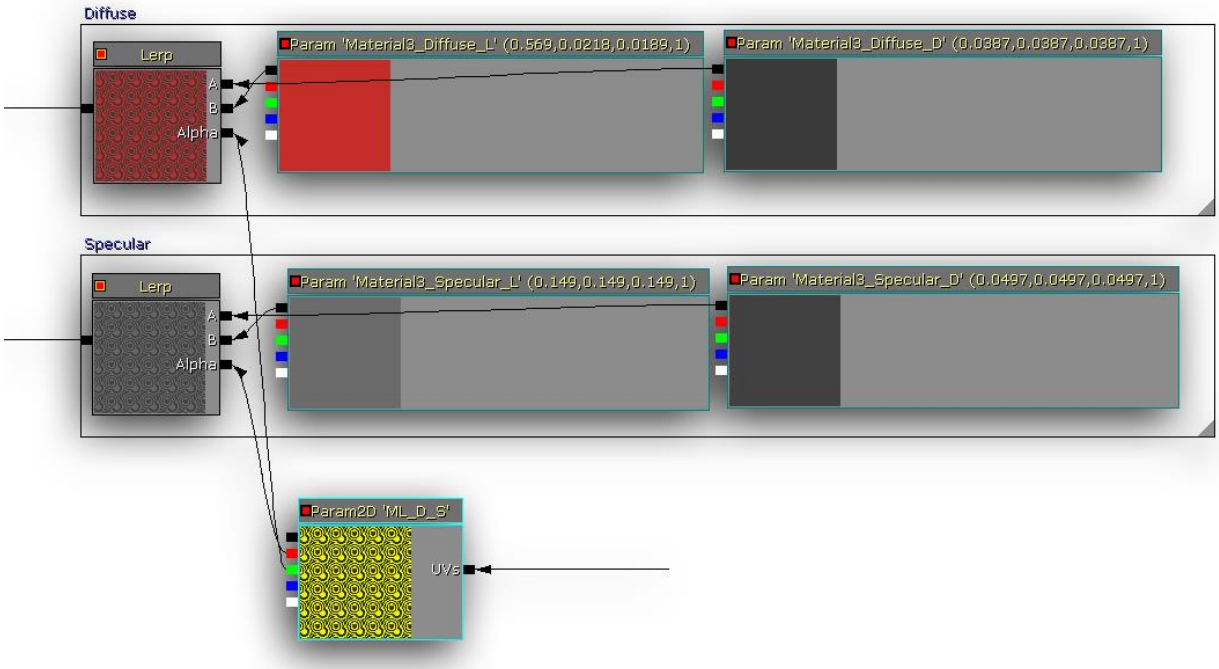
*Kuva 17. Toistuva tekstuuri yksittäisenä kuvana (vas.), ja toistettuna molemmissa akseleissa (oik.). (pattern4u 2014.)*

UDK:n materiaalieditorin tekstuurimäärärajoitusten takia täytyi kehittää toimiva systeemi, joka mahdollistaisi useiden tekstuurien tuonnin editoriin. Vakiona materiaalieditoriin voidaan tuoda ja ladata 12 eri tekstuuria. Määrä saattaa kuulostaa suurelta, mutta pelkästään 3d-objektin omat spesifiset kartat vievät jo vähintään 3 tekstuuripaikkaa mukaan luettuna ambient occlusion-kartta, normaalikartta ja värialuekartta. Näiden lisäksi pitäisi saada vielä kaikkiin kuuteen värialuekartan alueisiin oma toistuva diffuusikartta, spekulaarikartta ja normaalikartta. Kaikki toistuvat tekstuurikartat yhdessä erillisinä kuvatiedostoina vaatisivat 18 tekstuuripaikkaa, ja lisättynä 3d-objektin omiin karttoihin vaadittaisiin yhteensä 21 tekstuuripaikkaa.

Ideana oli, että yhteen tekstuuriin voitaisiin pakata diffuusi- ja spekulaarikartat mustavalkoisina kuvina omille värikanavilleen. Tämän lisäksi yhteen kuvaan voidaan pakata useampi diffuusi- ja spekulaarikartta rinnakkain, jolloin saadaan tekstuurin resoluutiosta riippuen hyvinkin laaja toistuvien tekstuurien kirjasto [Kuva 18].

*Kuva 18. Ylemmässä kuvassa yhteen tekstuuriin koodattu kaksi kuvaa yhteen, ja alapuolella normaalikartta toistuville tekstuureille. Normaalikartan tekstuurit ovat vajaita, koska niitä ei juurikaan käytetä esimerkeissä.*

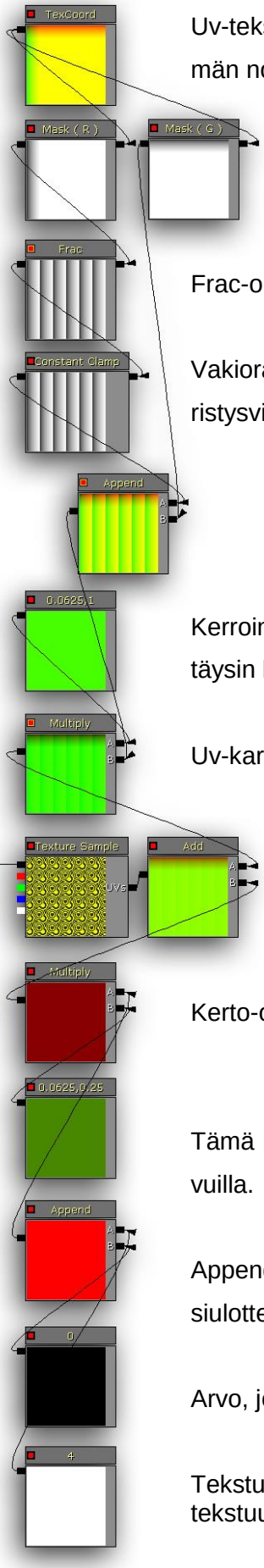
Tekstuurien mustavalkoisuus ei tässä tapauksessa juurikaan ollut haitaksi, koska mustavalkoista kuvaa voidaan käyttää maskina kahdelle halutulle värille materiaalieditorissa käyttäen Linear Interpolate (lerp) -operaatiota [kuva 19].




*Kuva 19. Vihreästä kanavasta saadaan toistuva diffuusikartta ja punaisesta kanavasta toistuva spekulaarikartta. Nämä kanavat ovat vakiona mustavalkoisia, mutta voidaan muuttaa värillisiksi Lerp-operaatiolla.*

Kirjastotekstuurissa yksittäisten toistuvien tekstuurien koko on 256*256 pikseliä ja koko kirjastotekstuurin leveys on 4096 pikseliä. Tämä antaa mahdollisuuden 16 eri diffuusitekstuurin ja 16 eri spekulaaritekstuurin tallentamiseen samaan tiedostoon. UDK tukee jopa 8192*8192 -resoluutioisia tekstuureja, jotka antaisivat mahdollisuuden käyttää kirjastotekstuurissa satoja uniikkeja tekstuureja.

Tekstuurien jaottelu ja toistaminen materiaalieditorilla tapahtuu kokonaan manipuloimalla uv-tekstuurikoordinaatteja matemaattisilla operaatioilla ja on toteutettu oheisella menetelmällä.




Uv-tekstuurikoordinaatti, jonka mukaisesti tekstuuri toistetaan kappaleen pintaan. Tämän noodin arvot määrittelevät tekstuurin koon x ja y -akseleissa.

Tekstuurikoordinaatin ’U’ ja ’V’ -akseleiden erottelu värimaskioperaatioilla. R- (red) ja G (green) -kanavat merkkaavat tekstuurin U ja V -koordinaatit yksiulotteisilla liukuväreillä.

Frac-operaatio toistaa x -akselin käyttämättömään data-alueeseen samaa kuvaa.

Vakiorajoitinoperaatiolla voidaan poistaa toistuvan tekstuurin saumakohtiin syntyviä pyöristysvirheitä.

Jaotellut U ja V -kartat yhdistetään takaisin kokonaiseksi uv-kartaksi.

Kerroin, jonka mukaan toistuva tekstuuri skaalataan u ja v -akseleissa. Tämä numero on täysin kirjastotekstuurin resoluutiosta riippuva ja täytyy asettaa suhteiltaan oikeaksi.

Uv-kartan kertominen tekstuuriresoluutioon suhteutetun arvon kanssa.

Uv-kartan ja tekstuurisiirtäjän yhdistäminen toimivaksi uv-kartta toistajaohjelmaksi.

Kerto-operaatio, joka yhdistää parametrit käytettävään formaattiin.

Tämä kaksivektoriarvo toimii kertoimena, jotta tekstuurialueita voidaan muuttaa tasaluvuilla.

Append -operaatiolla pystytään yhdistämään kahdesta yksiulotteisesta arvosta yksi kaksiulotteinen arvo.

Arvo, jolla voidaan muuttaa tekstuurialueen valintaa V -akselissa.

Tekstuurin valinnassa käytettävä tasaluku, jota muuttamalla nollan ja 15 välillä voidaan tekstuurialueen valintaa siirtämään tekstuurin U -akselissa.

## 4.4 Varjostinohjelmien osuuksien yhdistäminen lopulliseksi materiaaliksi

Tekstuurin toistajaohjelma voidaan kytkeä suoraan käytettävään kirjastotekstuuriin ja tästä erotella toistuvaksi diffusi- ja spekulaarikartaksi. Nämä tekstuurit voidaan asettaa näkymään maskialueilla, jotka tuotetaan värialuekarttaohjelmalla, josta sitten kaikki alueet voidaan yhdistää kahdeksi tekstuurikokonaisuudeksi, diffuusi- ja spekulaarikartaksi. Samalla menetelmällä voidaan jaotella ja yhdistää normaalikarttakirjastosta vastaavat alueet diffuusi- ja spekulaarikartan tueksi ja näin luoda toimiva kolmen tekstuurin kokonaisuus.

Diffuusi-, spekulaari-, ja normaalikartan yhdistäminen antaa jo melko hyvän näköisen lopputuloksen, mutta päätin lisätä lopputulokseen hieman lisää efektejä. Aikaisempien tekstuurien lisäksi kehitin varjostinohjelmaan kiiltäviä pintoja simuloivat heijastukset ja taustavaloa simuloivan ”rim light” -efektin [Kuva 20].

*Kuva 20. Varjostimien vaikutus mallin pinnalla. Vasemmalla malli ilman heijastus- ja taustavalovarjostinta, keskellä pelkästään taustavalovarjostin ja oikealla molemmat samanaikaisesti. Kuvan efektejä on korostettu havainnollistamisen vuoksi.*

Näiden ylimääräisten efektien lisääminen tuo mallin pinnalle mielestäni todella paljon lisää uskottavuutta reaaliaikanäkymässä. Varsinkin metallisten ja kiiltävien pintojen renderöinnissä nämä efektit toimivat erittäin hyvin.

## 4.5 Dynaamisesti muokattavat parametriset noodit ja materiaali-instanssit

UDK:n materiaalieditorissa luodut materiaalit voidaan instansoida helpommin muokattaviksi materiaaleiksi hyödyntäen parametrisia arvoja alkuperäisessä materiaalissa. Nämä parametriset arvot ilmaantuvat listaksi instansoidun materiaalin sisälle. Lähes kaikki arvot voidaan asettaa parametrisiksi, jolloin saadaan äärimmäisen joustava ja monikäyttöinen materiaali. Materiaali-instansseja voidaan luoda lähes rajaton määrä, ja kaikkiin instansseihin voidaan asettaa ainutlaatuiset arvot jokaiseen dynaamiseen parametriin. Näitä arvoja ovat esimerkissäni esimerkiksi kaikkien tekstuurien diffuusi-, ja spekulaarikarttojen kirkkaat ja tummat värit sekä niiden käyttämät mustavalkoiset kuviolliset tekstuurit ja normaalikartat.

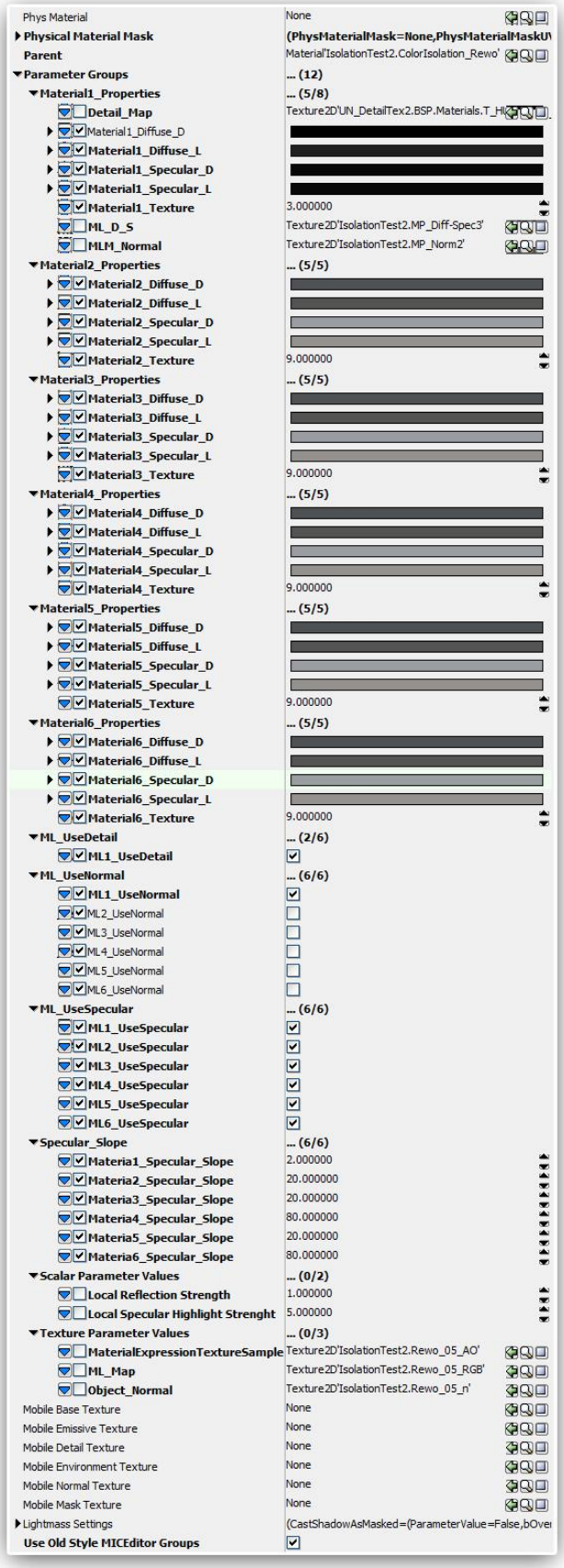
*Kuva 21. Instanssiparametreilla muokatut väripinnat. Vasemmalla pelkästään kirkkailla väreillä täytetyt alueet ja tekstuureilla täytetyt pinnat oikealla.*

Materiaali-instanssit antavat äärimmäisen nopean ja helpon tavan teksturoida lähes rajaton määrä 3d-malleja ainutlaatuisilla tekstuureilla. Alkuperäisen materiaalin muuttaminen päivittyy kaikkiin instansseihin välittömästi ja pitää näin kaikki instanssit täysin hallittavissa. Uusien parametrien lisäys ei myöskään aiheuta ongelmia instansseissa, vaan ne päivittyvät parametrilistaan automaattisesti.

*Kuva 22. Materiaali-instanssien parametreilla toteutettuja materiaalivariaatioita.*

Parametrieditori mahdollistaa materiaalien instanssien eri arvojen muokkaamisen nopeasti. Parametriarvojen muokkaus onnistuu myös suoraan pelimoottorin koodissa.

Materiaali-instanssien käyttäminen pelimoottorissa on usein paljon kevyempää, koska moottorin ei tarvitse pitää muistissa montaa erillistä materiaalikompositiota. Instansoidut materiaalit jakavat kaikki parametriset arvot, mutta pitävät ne muokattavissa.



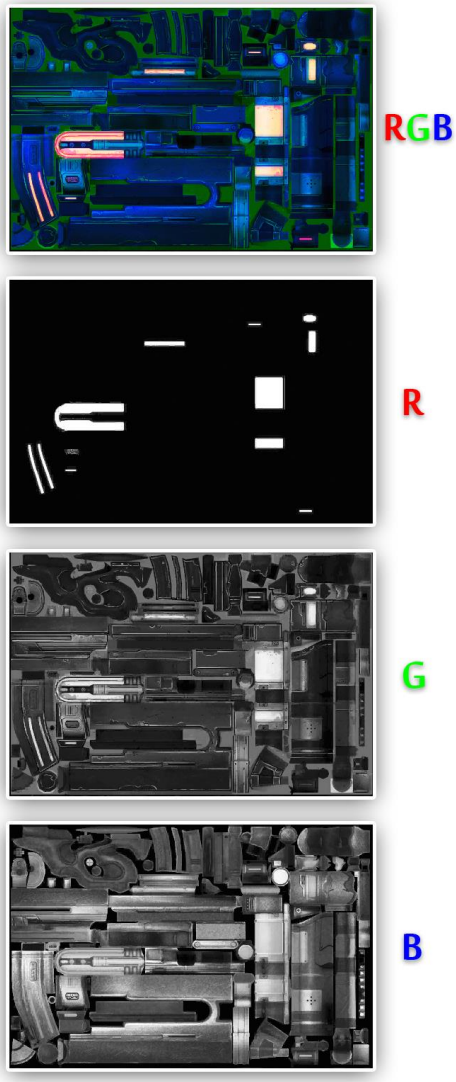

*Kuva 23. Näkymä instansoidun materiaalin kaikista eri parametriasetuksista.*

# 5 Vertailua Borderlandsin aseiden teksturointimenetelmään

Gearboxin Borderlands pelit ovat tunnettuja monelta osin niissä esiintyvien aseiden vuoksi. Pelissä esiintyvien aseiden määrä on lähes rajaton. Aseiden variaatioita voi olla miljoonia, joista jokainen on ainutlaatuinen. Tämän mahdollistaa asetyyppien palasten modulaarisuus.

Vaikka Borderlandsin kehittäjät eivät suoranaisesti ole tukeneet pelinsä modifioimista, on silti pieni määrä ihmisiä, jotka ovat tutkineet pelin sisältöä, ja näin saaneet selville, kuinka peliä on rakennettu.

Tekstuureja tutkimalla sain melko hyvän kuvan aseiden teksturointimenetelmästä. 3d-objektien spesifiset diffuusi-, spekulaari- ja emissiokartat on tallennettu yhteen tekstuuriin [Kuva 24] (S6, Polycount 2014). Tämän lisäksi malleissa käytetään myös normaalikarttaa ja kahta mahdollista ”overlay” -karttaa. Overlay-tekstuurikartoilla voidaan asettaa erillisiä kuvioita, tekstejä tai väriteemoja mallin päälle (S6, Polycount 2014).




*Kuva 24. Tekstuurin eri värikanaville tallennetut kartat. (polycount 2014.)*

Borderlandsin optimoinnin taso on näkyvillä lähes kaikissa pakatuissa tekstuureissa pelin sisällä. Liiallinen tekstuurien pakkaus aiheuttaa usein näkyviä visuaalisia ongelmia mallien päällä. Näistä visuaalisista ongelmista sainkin lisää osviittaa siihen, että kuinka pelin materiaalisysteemit on kasattu.

*Kuva 25. Tekstuurien pakkauksesta johtuvat artefaktit voivat näyttää 3d-mallien päällä erittäin huonolta. Hehkuvien osien pyöreät muodot kuvan vasemmalla jäävät epäselviksi. (borderlands.wikia 2014a.)*

Overlay tekstuureilla on saatu aseiden osat näyttämään erilaisilta ja toisistaan poikkeavilta. Jokaisesta aseen osasta on tehty viisi eri 3d-mallivariaatiota joiden tekstuurit ovat kaikki yhdessä suuressa uv-tekstuurikartassa. Näin eri osia ladattaessa ei tarvitse ladata kuin muutama tekstuuri: tekstuurin, joka sisältää diffuusi-, spekulaari- ja emissiokartan, tekstuurin, joka sisältää normaalikartan ja overlay-tekstuurit, joilla mallien väritys tehdään. Jos ruudulla on useampi variaatio samasta asetyypistä, silloin mallit jakavat pelkästään diffuusi-, spekulaari-, emissio- ja normaalikartat. Väritykseen vaadittavat overlay-kartat täytyy ladata erikseen, jos aseiden variaatiot eivät käytä samoja väriteemoja. Useamman eri väriteeman yhtäaikainen lataaminen voi viedä huomattavasti enemmän muistia.

Jos väriteemojen rakentamiseen käytettäisiin kehittämääni värialuekarttaa ja siihen liittyvää varjostinohjelmaa, voitaisiin säästyä ylimääräisiltä tekstuurien lataamisilta, ja näin mahdollisesti säästää muistia.

Systeemini käyttäminen on todennäköisesti raskaampi yksittäisen 3d-mallin teksturoimisessa, mutta teoriassa voisi olla huomattavasti kevyempi, ja vähemmän muistia vievä, kun teksturoitavia variaatioita on monia. Ladattavien tekstuurien määrä ei nouse, vaikka aseen variaatioita olisi satoja. Toisaalta, jos erinäköisiä 3d-malleja on monia, niihin täytyy silti ladata niiden spesifiset tekstuurit erikseen. Tässä kuitenkin voidaan säästää käyttämällä toistuvien tekstuurien kohdalla yhtä materiaalikirjastoa.


*Kuva 26. Pelissä esiintyvien rynnäkkökiväärien väriteemat valmistajineen. Teemojen määrittämiseen voidaan vaatia useita kymmeniä tekstuureja. (borderlands.wikia 2014b.)*

# 6 Pohdintaa

Vaikka materiaalisysteemini ei olekaan raakakoodin tasolla täysin optimoitu, antaa se silti uuden tavan teksturoida yksittäisiä 3d-malleja erittäin joustavasti suoraan pelimoottorin sisällä. Vastaavaan tekniikkaan en ole henkilökohtaisesti törmännyt työtä tehdessäni tai sitä ennen. Pääsin mielestäni haluttuun lopputulokseen. Sain kehitettyä menetelmän, joka ei ehkä pysty kokonaan korvaamaan nykypäivän teksturointitekniikoita, mutta voisi teoriassa olla erittäin hyvä työkalu suhteellisen yksinkertaisten 3d-mallien teksturoimisessa.

Jatkotutkimusta voisi tehdä esimerkiksi maskitekstuurin alpha-kanavan käyttämisessä. Esimerkiksi Png-kuvissa on vakiona neljä eri kanavaa, joista kolme ovat R, G ja B, ja neljäs on läpinäkyvyys. Koska UDK:n materiaalieditorissa voidaan tuodusta tekstuurista erotella eri kanavat helposti, antaisi neljännen kanavan värierottelu mahdollisuuden kahdelle ylimääräiselle maskitekstuurille, jolloin yhdestä png-kuvatiedostosta saataisiin ulos jopa kahdeksan uniikkia maskia tekstuuripintojen määrittämiselle.

Myös optimointia voisi jatkaa pidemmälle yhdistäen 3d-objektien spesifisiä normaalikarttoja ja ambient occlusion-karttoja samaan kuvatiedostoon. Normaalikartan alpha-kanavalle voidaan syöttää mustavalkoisena ambient occlusion-kartta, joka pystytään materiaalieditorissa erottelemaan ja asettamaan paikoilleen.

Verrattuna perinteisiin teksturointitekniikkoihin esittämässäni menetelmässä 3d-mallien tekstuureita voidaan muokata, ja vaihtaa lennossa, jopa pelimoottorin sisällä reaaliaikaisesti. Tämän päivän pelimallien tekstuureita ei pystytä pelimoottorin sisällä juurikaan muokkaamaan. Jos halutaan rinnakkaisille malleille useampi variaatio pinnalle laitettavasta tekstuurista, joudutaan yleensä tekemään täysin uusi tekstuuri mallille, joka saattaa viedä enemmän tekstuurimuistia ja lisää pelimoottorin kuormitusta.

## Lähteet


Booktype 2014. Tekstuurit Blenderissä. http://books.okf.fi/blender/tekstu-rointi/#.U1QPNfmSyxY. luettu 15.4.2014

borderlands.wikia 2014a. Material Grade. http://borderlands.wikia.com/wiki/Mate-rial_Grade. luettu 19.4.2014

borderlands.wikia 2014b. Construction. http://borderlands.wikia.com/wiki/Border-lands_2_Weapons. luettu 19.4.2014

Pattern4u 2014. Crop Circles. http://pattern4u.tk/565-crop-circles-seamless-texture. luettu 20.4.2014

Käyttäjä s6, Polycount 2014. Modular weapon pipeline. http://www.polycount.com/fo-rum/showthread.php?t=120670. luettu 20.4.2014

Wikipedia 2014a. Models. http://en.wikipedia.org/wiki/3D_modeling. luettu 14.4.2014

Wikipedia 2014b. Models. http://en.wikipedia.org/wiki/3D_modeling. luettu 14.4.2014

Wikipedia 2014c. Modeling process. http://en.wikipedia.org/wiki/3D_modeling. luettu 14.4.2014

Wikipedia 2014d. UV mapping. http://en.wikipedia.org/wiki/UV_mapping. luettu 14.4.2014

Wikipedia 2014e. Shader. http://en.wikipedia.org/wiki/Shader. luettu 16.4.2014

Wikipedia 2014f. Blender. http://en.wikipedia.org/wiki/Blender_(software). luettu 17.4.2014

Wikipedia 2014g. Adobe Photoshop. http://fi.wikipedia.org/wiki/Adobe_Photoshop. luettu 17.4.2014

Wikipedia 2014h. Unreal Engine. http://en.wikipedia.org/wiki/Unreal_Engine. luettu 18.4.2014

# Liitteet

## Liite 1. Color_Isolation.zip

UDK-pakettiitiedosto, (.upk) joka sisältää 3d-mallin, siihen liittyvät tekstuurit, materiaalit ja materiaali-instanssit. Mukana on myös värierotteluesimerkki, jossa havainnollistetaan värialueiden erottelu erikseen. Tiedoston avaaminen onnistuu ilmaisella Unreal Development Kitillä.